AMENDMENTS TO THE SPECIFICATION

Kindly amend paragraph [0020] as follows:

[0020] FIG. 1 provides the basic modules that are used in a spoken dialog system 100. A user 102 that is interacting with the system will speak a question or statement. An automatic speech recognition (ASR) module 104 will receive and process the sound from the speech. The speech is recognized and converted into text. AT&T's Watson ASR component is an example of such an ASR module. The text is transmitted to a spoken language understanding (SLU) module 106 (or natural language understanding (NLU) module) that determines the meaning of the speech, or determines the user's intent in the speech. This involves interpretation as well as decision: interpreting what task the caller wants performed and determining whether there is clearly a single, unambiguous task the caller is requesting - or, if not, determining actions that can be taken to resolve the ambiguity. The NLU 106 uses its language models to interpret what the caller said. The NLU processes the spoken language input wherein the concepts and other extracted data are transmitted (preferably in XML code) from the NLU 106 to the [[DM]] dialog manager (DM) application 108 along with a confidence score. The [[(DM)]] DM module 108 processes the received candidate intents or purposes of the user's speech and generates an appropriate response. In this regard, the DM 108 manages interaction with the caller, deciding how the system will respond to the caller. This is preferably a joint process of the DM engine 108 running on a Natural Language Services (NLS) platform (such as AT&T's infrastructure for NL services, for example) and the specific DM application 108 that it has loaded and launched. The DM engine 108 manages dialog with the caller by applying the compiled concepts returned from the NLU 106 to the logic models provided by the DM application 108. This determines how the system interacts with a caller, within the context of an ongoing dialog. The substance of the response is transmitted to a spoken language generation component (SLG) 110 which

generates words to be spoken to the caller 102. The words are transmitted to a text-to-speech module 112 that synthesizes audible speech that the user 102 receives and hears. The SLG 110 either plays back pre-recorded prompts or real-time synthesized text-to-speech (TTS). AT&T's Natural Voices ® TTS engine provides an example of a TTS engine that is preferably used. Various types of data and rules 114 are employed in the training and run-time operation of each of these components.

Kindly amend paragraph [0024] as follows:

[0024] Along with a dialog infrastructure, Florence offers tools to create a local development and test environment with many convenient and time-saving features to support dialog authoring. Florence also supplies a key runtime component for the VoiceTone Dialog Automation platform - the Florence ~~Dialog Manager (DM)~~ DM engine, an Enterprise Java Bean (EJB) on the VoiceTone/NLS J2EE application server. Once a DM application is deployed on a platform such as the VoiceTone platform, the DM engine uses the logic built into the application's dialogs to manage interactions with end-users within the context of an on-going dialog.

Kindly amend paragraph [0067] as follows:

[0067] The development system and method of the invention supports component-based development of complex dialog systems. This includes support for the creation and re-use of parameterized dialog components and the ability to retrieve values from these components using either local results or global variables. An example of the reusable components includes a subdialog that requests credit-card information. This mechanism for ~~re-usable~~ reusable dialog components pervades the entire system, providing a novel level of support for dialog authors.

The author can expect components to operate successfully with respect to the global parameters of the application. Examples of such global parameters comprise the output template and context shift parameters. The components can be used recursively within the system, to support recursive dialog flows if necessary. Therefore, while a subdialog is controlling the conversation, if a context shift occurs, the subdialog is isolated from the application dependencies (such as a specific piece of information that the application provides like the top selling books on amazon.com). Being isolated from the application dependencies allows for the subdialog to indicate a context shift and transfer control back to another module without trying to continue down a pre-determined dialog.

Kindly amend paragraph [0078] as follows:

[0078] When the DM author wishes to uses any of these patterns, the control structure is already in place so they can focus on the parameters of the structure that are specific to their application. Context shifts, for example, require a way to ensure that certain key phrases (such as "quit", "start over", or a complete change of topic) or conditions will trigger an application specific response, even when a pre-existing dialog definition is being ~~re-used~~ reused.